Machine Learning for NLP

Unsupervised Learning

Aurélie Herbelot

2018

Centre for Mind/Brain Sciences
University of Trento

Unsupervised learning

- In unsupervised learning, we learn without training data.
- The idea is to find a structure in the unlabeled data.
- The following unsupervised learning techniques are fundamental to NLP:
  - dimensionality reduction (e.g. PCA, using SVD or any other technique);
  - clustering;
  - *some* neural network architectures.

Dimensionality reduction

Dimensionality reduction

- Dimensionality reduction refers to a set of techniques used to reduce the number of variables in a model.
- For instance, we have seen that a count-based semantic space can be reduced from thousands of dimensions to a few hundreds:
  - We build a space from word co-occurrence, e.g. *cat - meow: 56* (we have seen *cat* next to *meow* 56 times in our corpus.
  - A complete semantic space for a given corpus would be a $N \times N$ matrix, where N is the size of the vocabulary.
  - N could be well in the hundreds of thousands of dimensions.
  - We typically reduce N to 300-400.

From PCA to SVD

- We have seen that Principal Component Analysis (PCA) is used in the Partial Least Square Regression algorithm for *supervised learning*.
- PCA is *unsupervised* in that it finds 'the most important' dimensions in the data just by finding structure in that data.
- A possible way to find the principal components in PCA is to perform Singular Value Decomposition (SVD).
- Understanding SVD gives an insight into the nature of the principal components.

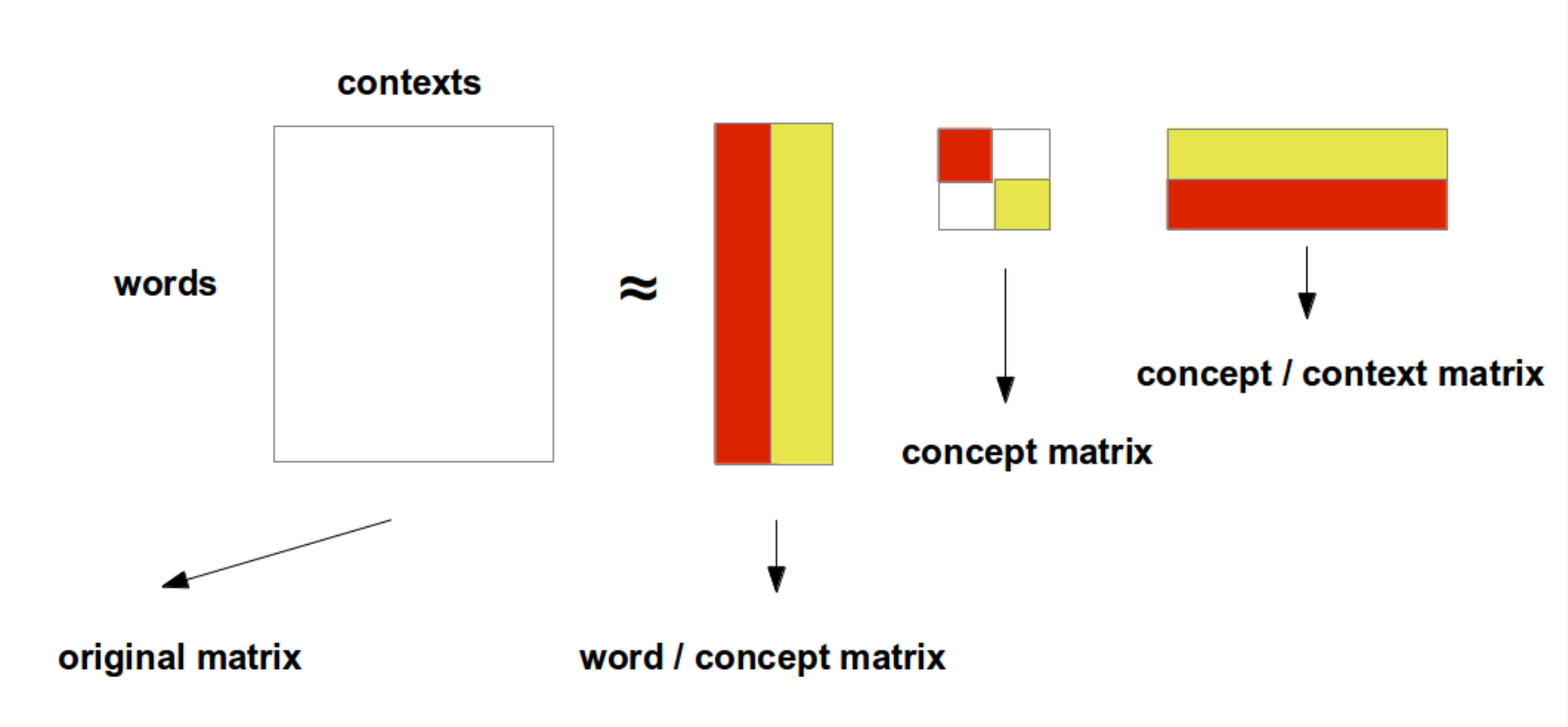
Singular Value Decomposition

- SVD is a matrix factorisation method which expresses a matrix in terms of three other matrices:

$$A = U\Sigma V^T$$

- U and V are orthogonal: they are matrices such that
  - $UU^T = U^T U = I$
  - $VV^T = V^T V = I$
- Σ is a diagonal matrix (only the diagonal entries are non-zero).

Singular Value Decomposition over a semantic space



The SVD derivation

- From our definition, $A = U\Sigma V^T$, it follows that...
- $A^T = V\Sigma^T U^T$
- $A^T A = V\Sigma^T U^T U\Sigma V^T = V\Sigma^2 V^T$
  (Recall that $U^T U = I$ because U is orthogonal.)
- $A^T AV = V\Sigma^2$
  (The inverse V^{-1} of an orthogonal matrix is V^T, since $V^T V = I$.)
- Similarly, $AA^T U = U\Sigma^2$.

SVD and eigenvectors

- An *eigenvector* of a linear transformation is a non-zero vector that doesn't change its direction when that linear transformation is applied to it:

$$Av = \lambda v$$

  v is the eigenvector, *lambda* is the eigenvalue.

- Let's consider again the end of our derivation: $A^T AV = V\Sigma^2$.
- The columns of V are the eigenvectors of $A^T A$. (Similarly, the columns of U are the eigenvectors of AA^T.)

The singular values of SVD

- If V contains the eigenvectors of $A^T A$, Σ^2 encapsulates its eigenvalues:
  $A^T AV = V\Sigma^2$.
- Σ itself contains the square roots of the eigenvalues, also known as *singular values*.

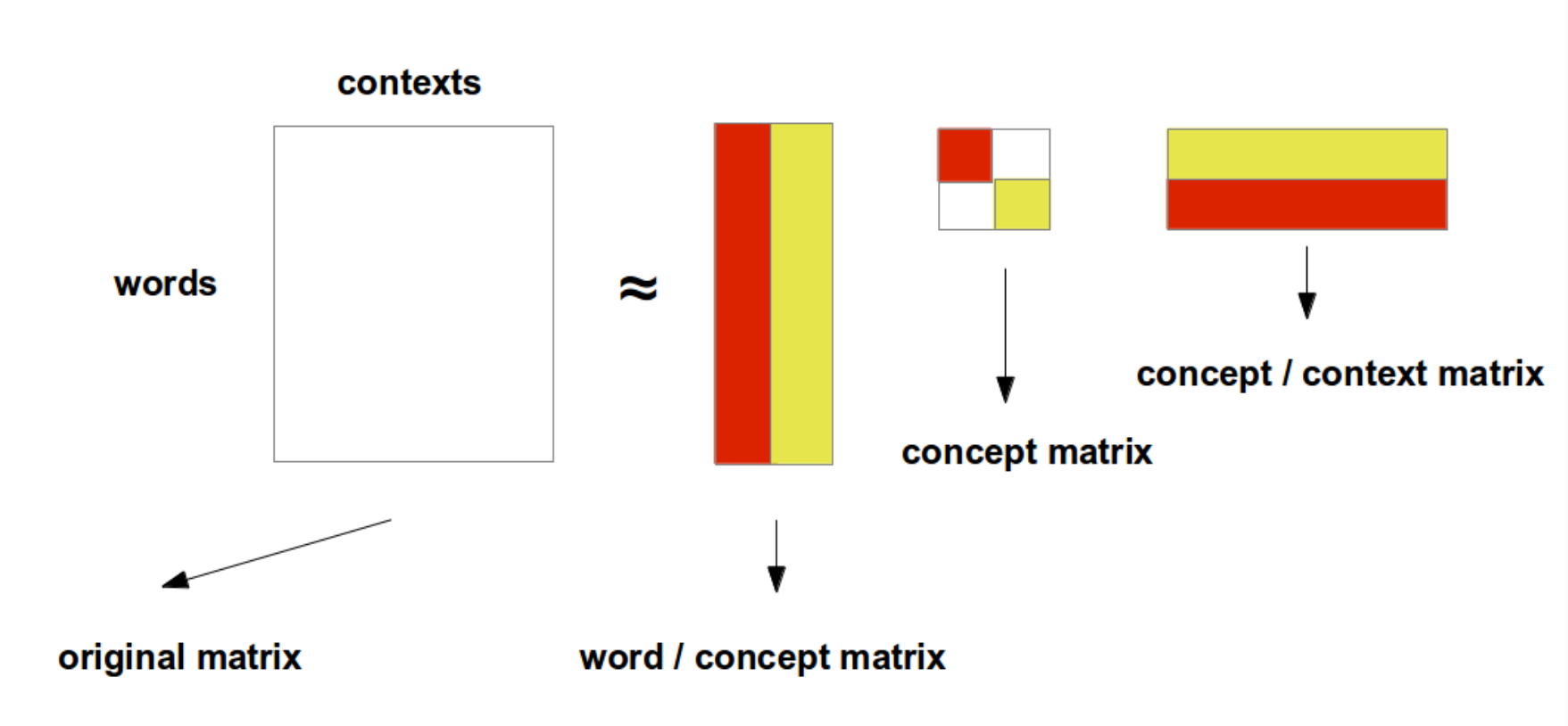
SVD at a glance

- Calculate $A^T A$.
- Calculate the eigenvalues of $A^T A$ and sort them in descending order. Take their square roots to obtain the singular values of $A^T A$ (i.e. the matrix Σ).
- Use the eigenvalues to compute the eigenvectors of $A^T A$. These eigenvectors become the columns of V.
- Compute $U = AV\Sigma^{-1}$.

Finally... dimensionality reduce!

- Now we know the value of U, Σ and V.
- To obtain a reduced representation of A, choose the top k singular values in Σ and multiply the corresponding columns in U by those values.
- Example: the original matrix is 10000 x 5000. The SVD produces:
  - U: 10000 x 10000
  - Σ: 10000 x 5000
  - V: 5000 x 5000
- We can now take the first 300 singular value of Σ, and the corresponding 300 columns of U. We are now multiplying a matrix of 10000 x 300 by a matrix of 300 x 300.

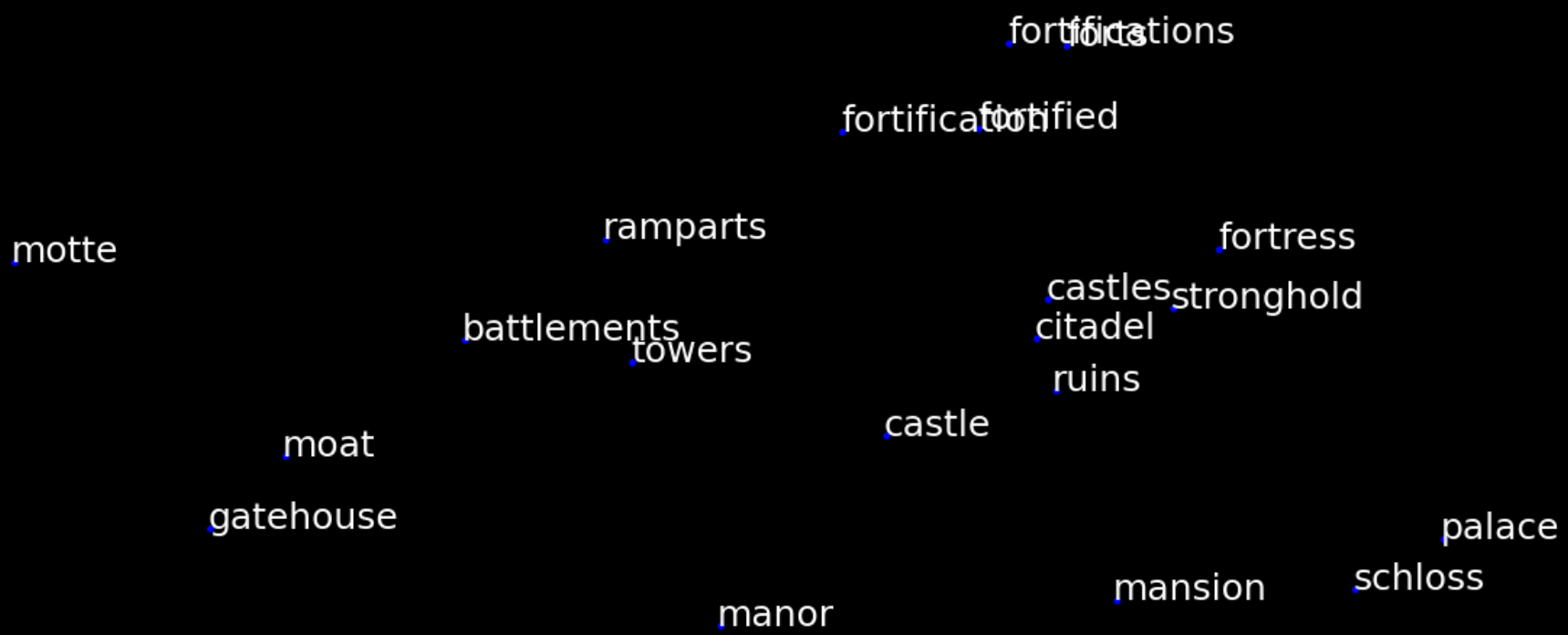
Singular Value Decomposition



What semantic space?

- Singular Value Decomposition (LSA – Landauer and Dumais, 1997). A new dimension might correspond to a generalisation over several of the original dimensions (e.g. the dimensions for *car* and *vehicle* are collapsed into one).
  - + Very efficient (200-500 dimensions). Captures generalisations in the data.
  - - SVD matrices are not straightforwardly interpretable.

SVD for visualisation



Dimensionality reduction and primitives

So what are those dimensions?

- Dimensionality-reduced spaces are not very interpretable.
- A count-based semantic space has dimensions labelled with words (or any other linguistic constituent). After reduction, the dimensions are unlabelled.
- The following is taken from Boleda & Erk (2015): *Distributional Semantic Features as Semantic Primitives - or not*.

Semantic primitives

- Word meaning can be represented in terms of primitives (Fodor et al, 1980):
  $man = [+\text{HUMAN}, +\text{MALE}]$

Category	Primes
Substantives	I, YOU, SOMEONE, PEOPLE, SOMETHING/THING, BODY
Relational Substantives	KIND, PART
Determiners	THIS, THE SAME, OTHER~ELSE~ANOTHER
Quantifiers	ONE, TWO, SOME, ALL, MUCH/MANY, LITTLE/FEW
Evaluators	GOOD, BAD
Descriptors	BIG, SMALL
Mental predicates	THINK, KNOW, WANT, DON'T WANT, FEEL, SEE, HEAR
Speech	SAY, WORDS, TRUE
Actions, Events, Movement	DO, HAPPEN, MOVE
Existence, Possession	BE (SOMEWHERE), THERE IS, BE (SOMEONE/SOMETHING), (IS) MINE
Life and Death	LIVE, DIE
Time	WHEN/TIME, NOW, BEFORE, AFTER, A LONG TIME, A SHORT TIME, FOR SOME TIME, MOMENT
Space	WHERE/PLACE, HERE, ABOVE, BELOW, FAR, NEAR, SIDE, INSIDE, TOUCH (CONTACT)
Logical Concepts	NOT, MAYBE, CAN, BECAUSE, IF
Intensifier, Augmentor	VERY, MORE
Similarity	LIKE/AS/WAY

Table adapted from Levisen and Waters 2017[2] and Goddard and Wierzbicka 2014 [1].

Semantic primitives: why

- To capture aspects of the real world (see again difference between word usage and extension).
- To formalise commonalities between near-synonymous expressions:
  Kim gave a book to Sandy $\approx$ *Sandy received a book from Kim.*
- To do inference: *John is a human* follows from *John is a man.*

Problems with primitives

- Primitives are ill-defined. They need to be extra-linguistic in order to avoid circularity.
  E.g. if BACHELOR is defined as $=$ [+MAN, +UNMARRIED], what are MAN and UNMARRIED?
- But sensory-motor properties are probably not fit to explain the semantic blocks of language (at least alone).
  E.g. GRANDMOTHER in *my cat's grandmother*.

Problems with primitives

- If primitives were real, they should be detectable in psycholinguistic experiments. For instance, the effect of negation should be noticeable in processing times:
  $\textsc{bachelor} = [+\textsc{man}, -\textsc{married}]$
  But no such effect has been found.
- Meaning nuances are lost in primitives:
  $\textsc{kill} \approx [+\textsc{cause}, -\textsc{alive}]$
  (There are many causes for not being alive which do not involve killing.)

Relation to distributional semantic spaces

- Might the features of a distributional space replace the notion of primitive?
  - Only a reduced number of dimensions is needed ($\approx$ 300 seems to be a magic number).
  - They can include both linguistic and extra-linguistic information.
  - Distributional representations do correlate with measurable human judgements.
  - They are made of continuous values, which in combination can express very graded meanings.

Inference

- Does distributional semantics satisfy the requirement that primitives should give us inference?
- To some extent...
  - Hyponymy can be learnt (e.g. Roller et al, 2014).
  - Some entailment relations can be learnt (e.g. Baroni et al 2012 on quantifiers.)
- But: distributional inference is *soft*, non-logical inference.

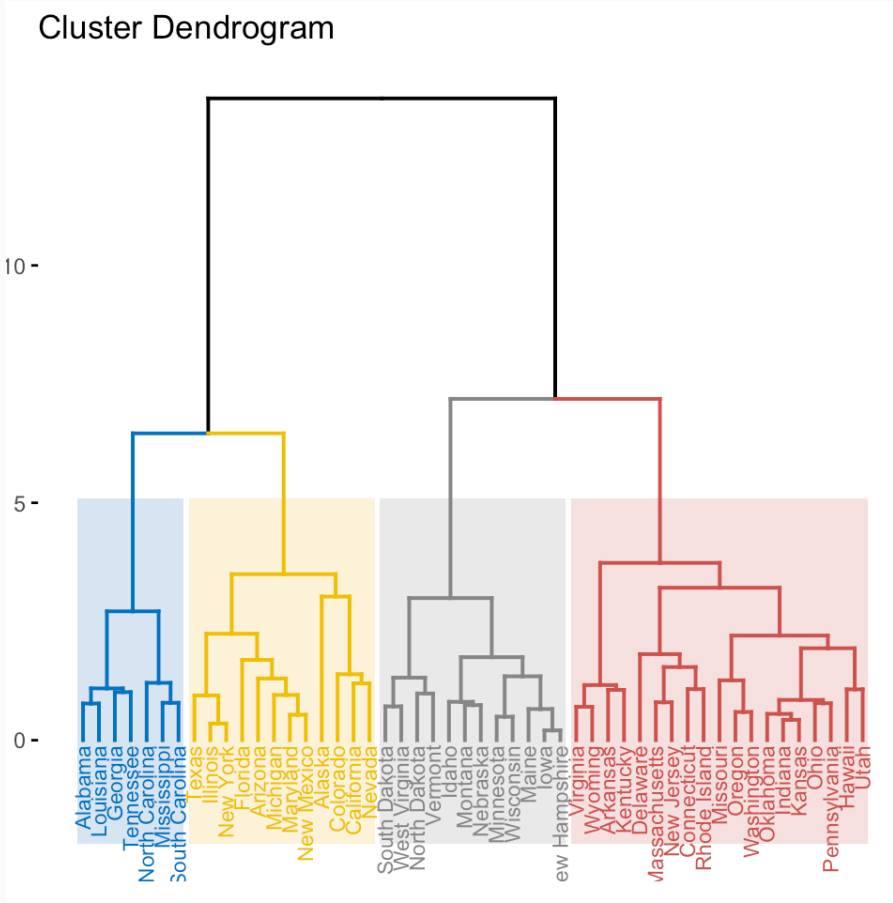
Dimensions as semantic primes?

- It may be that reduced distributional matrices capture important commonalities across word meanings and can be seen as an alternative to primitives.
- But we remain unable to tell what those primitives stand for.
- (We can hack it, though...)

Clustering

Clustering algorithms

- A clustering algorithm partitions some objects into groups named *clusters.*
- Objects that are similar according to a certain set of features should be in the same cluster.



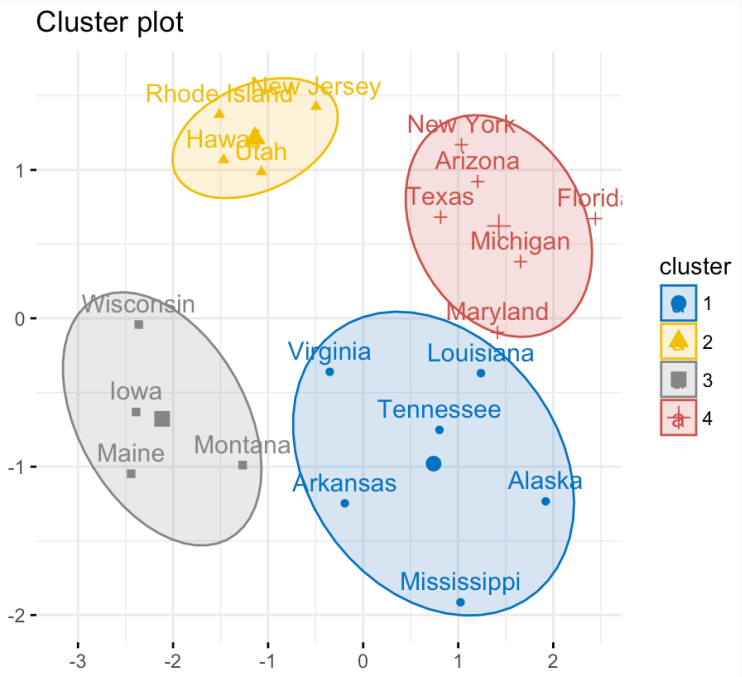
From http://www.sthda.com/english/articles/25-cluster-analysis-in-r-practical-guide/

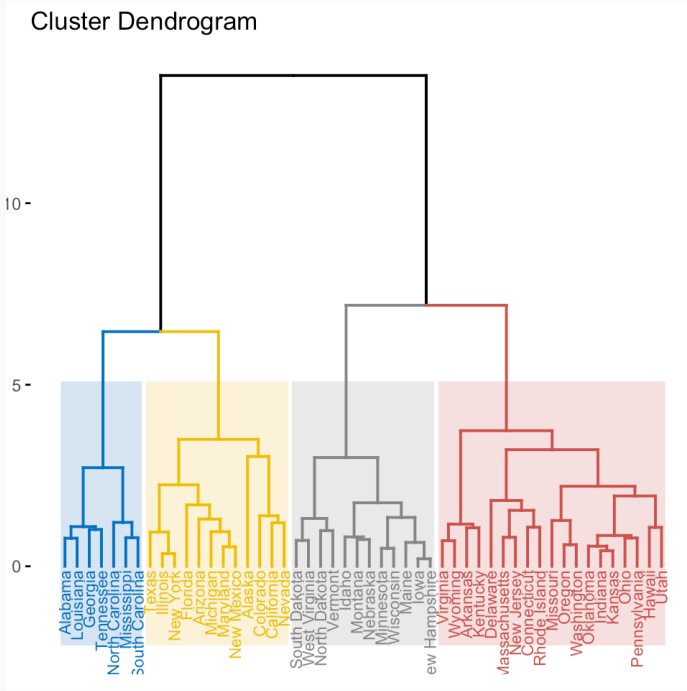
Why clustering

- **Example:**[1] we are translating from French to English, and we know from some training data that:
  - *Dimanche* $\longrightarrow$ *on Sunday*;
  - *Mercredi* $\longrightarrow$ *on Wednesday*;
- What might be the correct preposition to translate *Vendredi* into the English ___ *Friday*? (Given that we haven't seen it in the training data.)
- We can assume that the days of the week form a semantic cluster, which behave in the same way syntactically. Here, clustering helps us **generalise**.

[1] Example from Manning & Schütze, *Foundations of statistical language processing.*

Flat vs hierarchical clustering





Flat clustering

Hierarchical clustering

From http://www.sthda.com/english/articles/25-cluster-analysis-in-r-practical-guide/

Soft vs hard clustering

- In hard clustering, each object is assigned to only one cluster.
- In soft clustering, assignment can be to multiple clusters, or be probabilistic.
- In a *probabilistic* setup, each object has a probability distribution over clusters. $P(c_k|x_i)$ is the probability that object x_i belongs to cluster c_k.
- In a *vector space*, the degree of membership of an object x_i to each cluster can be defined by the similarity of x_i to some representative point in the cluster.

Centroids and medoids

- The *centroid* or *center of gravity* of a cluster c is the average of its N members:

$$\mu_k = \frac{1}{N} \sum_{x \in c_k} x$$

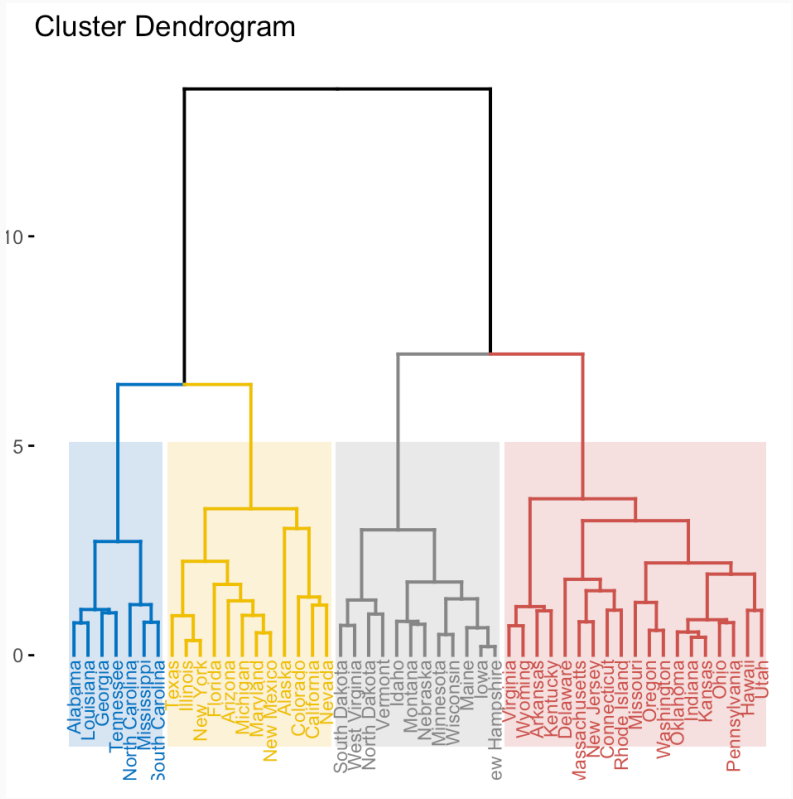
- The *medoid* of a cluster c is a prototypical member of that cluster (its average dissimilarity to all other objects in c is minimal):

$$x_{\text{medoid}} = \text{argmin}_{y \in \{x_1, x_2, \cdots, x_n\}} \sum_{i=1}^{n} d(y, x_i)$$

Hierarchical clustering: bottom-up

- Bottom-up agglomerative clustering is a form of hierarchical clustering.
- Let's have n datapoints $x_1 ... x_n$. The algorithm functions as follows:
  - Start with one cluster per datapoint: $c_i := \{x_i\}$.
  - Determine which two clusters are the most similar and merge them.
  - Repeat until we are left with only one cluster $C = \{x_1 ... x_n\}$.

Hierarchical clustering: bottom-up



NB: The order in which clusters are merged depends on the similarity distribution amongst datapoints.

Hierarchical clustering: top-down

- Top-down divising clustering is the pendent of agglomerative clustering.
- Let's have n datapoints again: $x_1 ... x_n$. The algorithm relies on a *coherence* and a *split* function:
  - Start with a single cluster $C = \{x_1 ... x_n\}$
  - Determine the least coherent cluster and split it into two new clusters.
  - Repeat until we have one cluster per datapoint: $c_i := \{x_i\}$.

Coherence

- **Coherence** is a measure of the pairwise similarity of a set of objects.
- A typical coherence function:

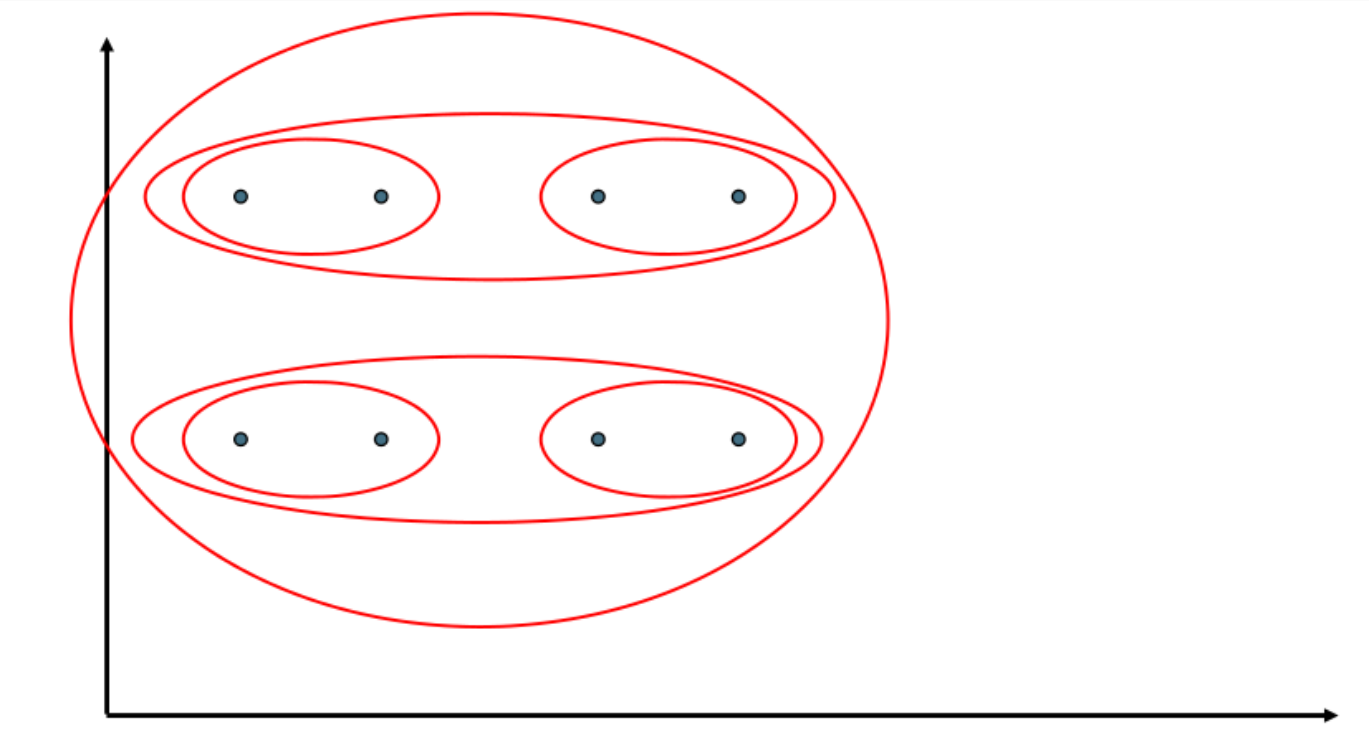
$$Coh(x_{1...n}) = mean\{Sim(x_i, x_j), ij \in 1...n, i < j\}$$

- E.g. coherence may be used to calculate the consistency of topics in topic modelling.

Similarity functions for clustering

- **Single link:** similarity of two most similar objects across clusters.
- **Complete link:** similarity of two least similar objects across clusters.
- **Group-average:** average similarity between objects. (Here, the average is over all pairs, within and across clusters.)

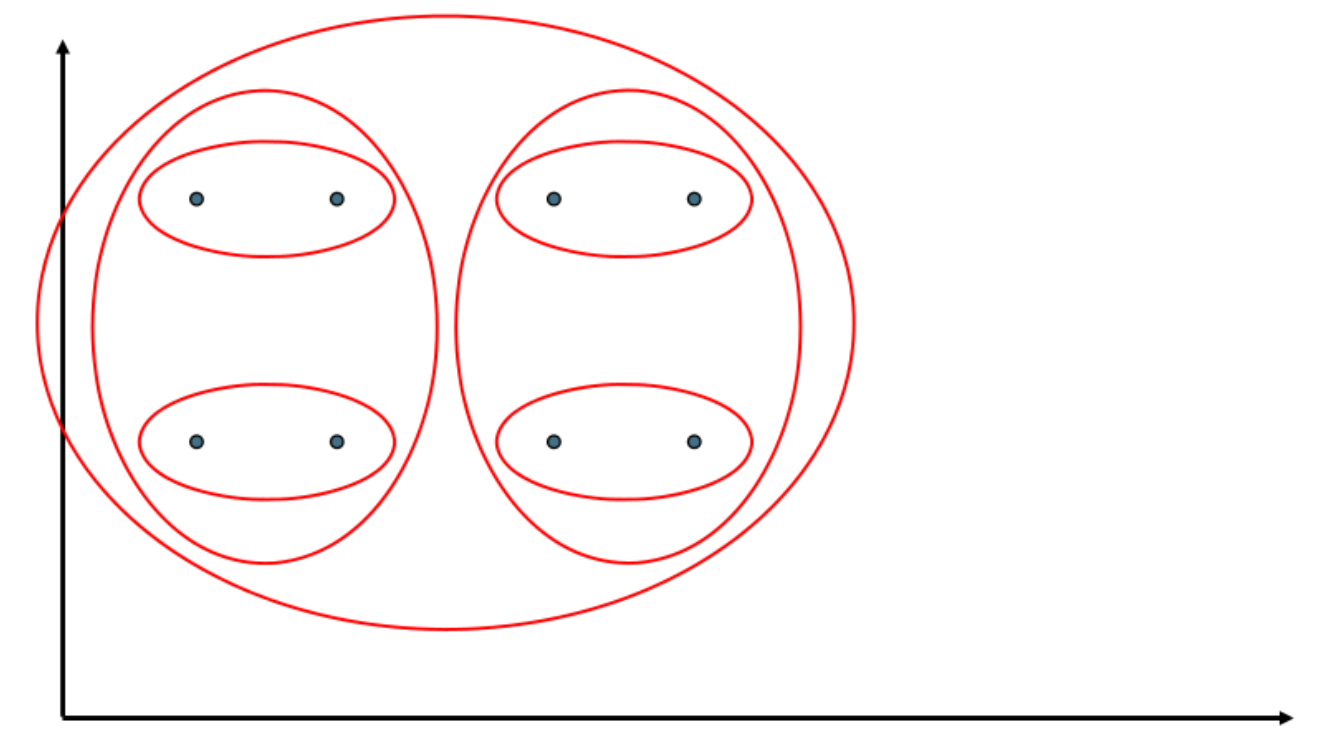
Effect of similarity function



Clustering with single link

Graph taken from Manning, Raghavan & Schütze:
http://www.cs.ucy.ac.cy/courses/EPL660/lectures/lecture12-clustering.pdf

Effect of similarity function



Clustering with complete link

Graph taken from Manning, Raghavan & Schütze:
http://www.cs.ucy.ac.cy/courses/EPL660/lectures/lecture12-clustering.pdf

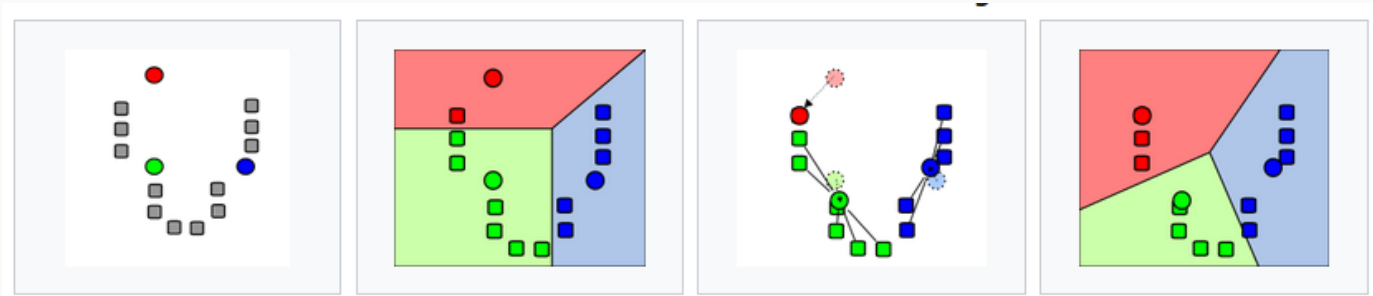
K-means clustering

- K-means is the main flat clustering algorithm.
- Goal in K-means: minimise the *residual sum of squares (RSS)* of objects to their cluster centroids:

$$RSS_k = \sum_{x \in c_k} |x - \mu_k|^2$$

- The intuition behind using RSS is that good clusters should have a) small intra-cluster distances; b) large inter-cluster distances.

K-means algorithm



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Source: https://en.wikipedia.org/wiki/K-means_clustering

Convergence

- Convergence happens when the RSS will not decrease anymore.
- It can be shown that K-means will converge to a local minimum, but not necessarily a global minimum.
- Results will vary depending on seed selection.

Initialisation

- Various heuristics to ensure good clustering:
  - exclude outliers from seed sets;
  - try multiple intialisations and retain the one with lowest RSS;
  - obtain seeds from another method (e.g. first do hierarchical clustering).

Number of clusters

- The number of clusters K is predefined.
- The ideal K will minimise variance within each cluster as well as minimise the number of clusters.
- There are various approaches to finding K. Examples that use techniques we have already learnt:
  - cross-validation;
  - PCA.

Number of clusters

- **Cross-validation:** split the data into random folds and cluster with some k on one fold. Repeat on the other folds. If points consistently get assigned to the same clusters (if membership is roughly the same across folds) then k is probably right.
- **PCA:** there is no systematic relation between principal components and clusters, but heuristically checking how many components account for most of the data's variance can give a fair idea of the ideal cluster number.

Evaluation of clustering

- Given a clustering task, we want to know whether the algorithm performs well on that task.
- E.g. clustering concepts in a distributional space: *cat* and *giraffe* under ANIMAL, *car* and *motorcycle* under VEHICLE (Almuhareb 2006)
- Evaluation in terms of 'purity': if all the concepts in one automatically-produced cluster are from the same category, purity is 100%.

Purity measure

- Given clustered data, purity is defined as

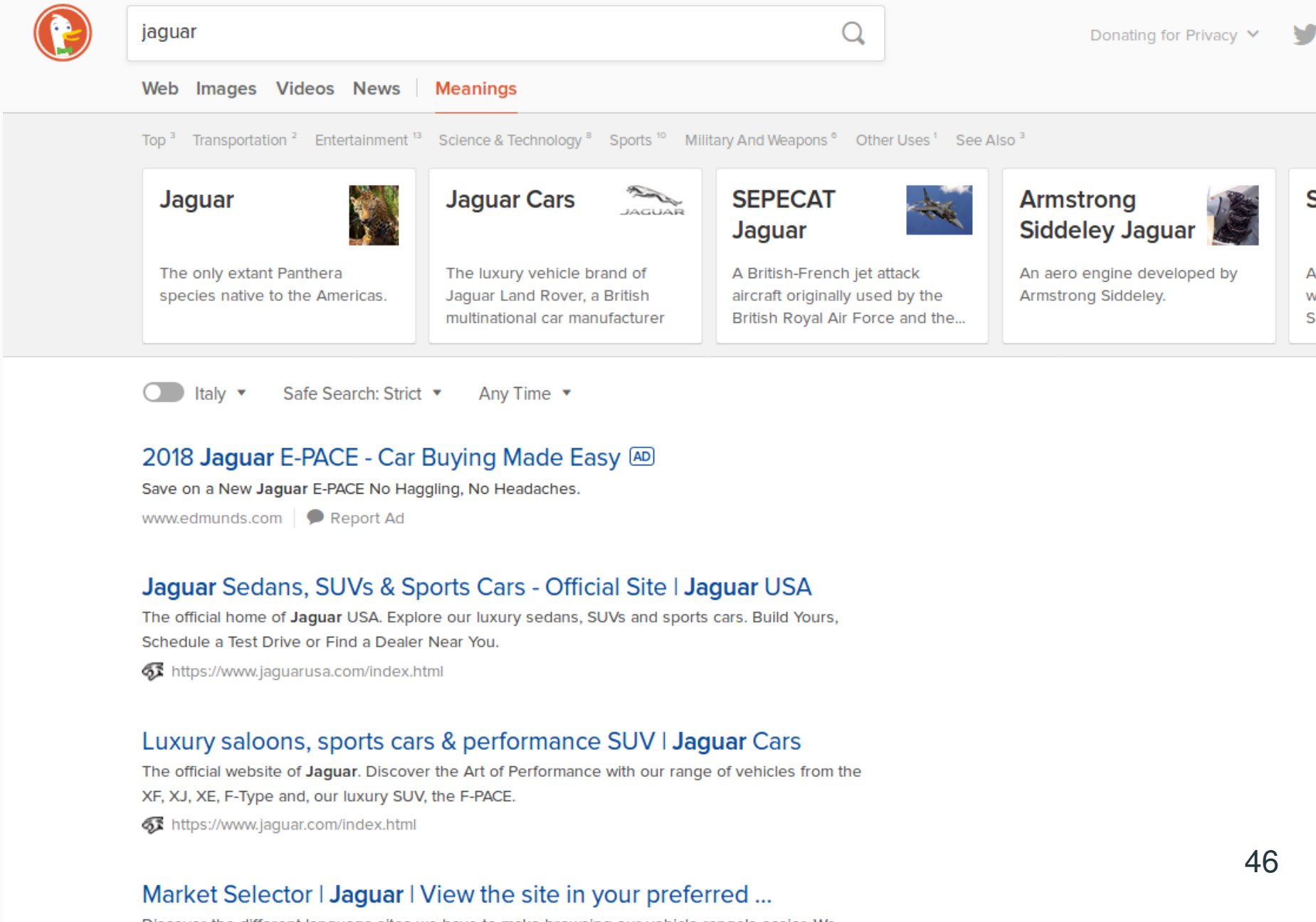
$$P(S_r) = \frac{1}{n_r} max_i(n_r^i)$$

- Example: given the cluster S_1 {A A A T A A A T T A}:

$$P(S_1) = \frac{1}{10} max(7, 3) = 0.7$$

- NB: here, the annotated data is only used for evaluation, *not for training!!* (Compare with k-NN algorithm.)

Clustering in real life



Weakly-supervised learning (bootstrapping)

Bootstrapping

- *Bootstrapping*, sometimes also referred to as *weakly-supervised learning* is machine learning from a small set of data.
- The idea behind such methods is that the algorithm can itself expand an initial ruleset.

Automatic ontology extraction

- Save person-years... and potentially save lives...
- Automatic fact extraction:
  - *X suppresses Y* (in a chemistry paper)
  - *people suffering from Z have high levels of Y* (in a medical journal)
  - $\longrightarrow$ X could help people with Z???
- If we could automatically extract large amounts of relations from naturally-occuring data, we could infer new facts...

Pattern-based extraction: Hearst (1992)

- Automatically extract hyponyms: Hearst noticed that some surface patterns indicate hyponymic relations with high probability.

(2) *such NP as {NP ,}* {(or | and)} NP*

`... works by such authors as Herrick, Goldsmith, and Shakespeare.`

$\Longrightarrow$ *hyponym("author", "Herrick"),*
hyponym("author", "Goldsmith"),
hyponym("author", "Shakespeare")

(3) *NP {, NP}* {,} or other NP*

`Bruises, wounds, broken bones or other injuries ...`

$\Longrightarrow$ *hyponym("bruise", "injury"),*
hyponym("wound", "injury"),
hyponym("broken bone", "injury")

Finding new patterns: the Snowball algorithm
Agichtein & Gravano (2000)

- Let's assume we want to find out a list of organisations and their headquarters.
- Let's also assume we already know some of them.

Organisation	Location
Microsoft	Redmond
Exxon	Irving
IBM	Armonk
Boeing	Seattle
Intel	Santa Clara

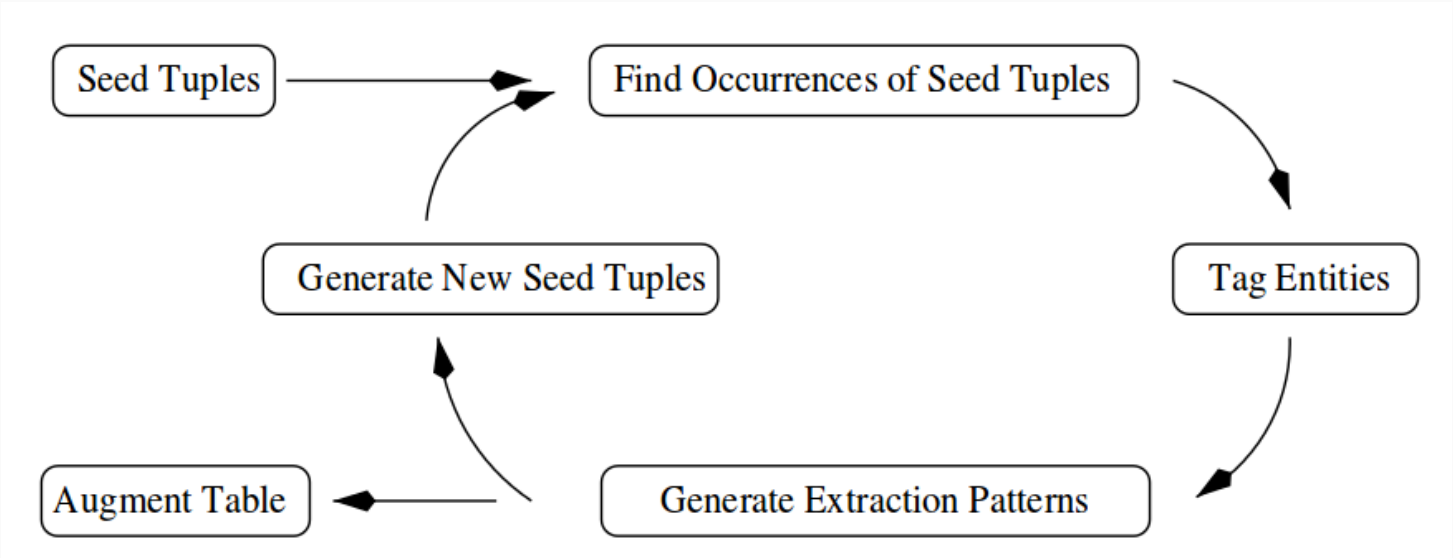
Finding new patterns: the Snowball algorithm
Agichtein & Gravano (2000)

- We can use already known relations as ‘seeds’ to discover new patterns.
- For instance, observe that the pair <Microsoft, Richmond> is found in the sentence fragment:
  *computer servers at **Microsoft**’s headquarters in **Richmond***.
- Build a pattern from observed contexts:
  <STRING1>’s headquarters in <STRING2>

Exploiting named entities

- A good pattern must be *reliable* and have good *coverage*.
- Assume we have found the following pattern:
  <STRING2>-based <STRING1>
  as in *Richmond-based Microsoft*
- Good pattern? It will return:
  - Seattle-based Boeing
  - Chicago-based company
  - ...
- We must make sure that STRING1 is an organisation and STRING2 a location! (At the very least do Named Entity Recognition.)

Overview of the Snowball algorithm



Snowball: measure of pattern confidence

- Let's assume we extracted the pattern *<ORGANISATION , LOCATION>*. How reliable is it?
- Using our existing set of seeds, the confidence of a pattern is given by:

$$Conf(P) = \frac{P.positive}{P.positive + P.negative}$$

- Example:
  - **Exxon, Irving**, said
  - **Intel, Santa Clara**, cut prices
  - invest in **Microsoft, New York**-based analyst Jane Smith said

$Conf(P) = \frac{2}{2+1} = 0.67$

Snowball: Measure of tuple confidence

- Let's assume that $Conf(P)$ is a rough approximation of the probability of P to produce good tuples.
- We can calculate a tuple T's confidence as a function of the confidence of the patterns that produce it:

$$Conf(T) = 1 - \prod_{i=0}^{|P|}(1 - Conf(P_i)) \quad (1)$$

Snowball: summary

- Snowball is a way to greedily acquire new information, using a pattern-based approach.
- Danger: increasing recall by producing many new patterns can be detrimental to precision.
- The notion of confidence is key to the performance of the system!